Architecture, Philosophy and Performance of JPIP: Internet Protocol Standard for JPEG2000

David Taubman[a] and Robert Prandolini[b]

[a]School of Electrical Engineering and Telecommunications, The University of New South Wales
[b]Defence Science and Technology Organisation, Department of Defence, Australia

ABSTRACT

JPEG2000 is a family of technologies based on the image compression system defined in IS 15444-1. Presently, the ISO/IEC Joint Technical Committee of Photographic Experts (JPEG) is developing an international standard for interactivity with JPEG2000 files, called JPIP; it will become Part 9 of the standard. One of the main goals of JPIP is to exploit the multi-resolution and spatially random access properties of JPEG2000, to permit "smart dissemination" of the data for client-server based applications. The purpose of this paper is to discuss the principles and philosophy of operation underlying the JPIP standard, to outline aspects of the architecture of JPIP systems, and to report on the performance of a prototype implementation.

Keywords: JPEG2000, interactive imaging, internet protocols, image retrieval.

1. INTRODUCTION

The ISO/IEC Joint Technical Committee of Photographic Experts (JPEG) is currently developing an international standard for interactivity with JPEG2000 code-streams and files. The work item is known as JPIP, and will become Part 9 of the JPEG2000 standard. The purpose of JPIP is to standardize a means of interacting with JPEG2000 based data in an efficient and effective manner.

The JPEG2000 image compression standard offers many desirable features in support of interactive access to large images. Chief amongst these are resolution scalability, progressive refinement (or quality scalability), spatial random access, and highly efficient compression. However, the compression standard itself describes only a code-stream syntax, suitable for storing the compressed data in a file. One way to interact remotely with the image content is for an intelligent browsing client to access appropriate byte ranges from the file. Such an approach allows existing HTTP servers to be used as-is, exploiting the support offered by HTTP/1.1 for byte range accesses. Indeed, such an approach was proposed by Depande and Zeng[1]. However, this approach requires the inclusion of index tables which an intelligent client can read to determine the locations (byte ranges) of the relevant compressed data and header information. JPIP standardizes the form of the index tables to be used for such purposes, as discussed in Section 5.

While recognizing the usefulness of a byte ranging approach, the JPIP standard is primarily concerned with the description of a new protocol for interacting with JPEG2000 content. With the JPIP protocol, a client does not directly access the compressed file. Rather, it formulates requests using a simple descriptive syntax which identifies the current "focus window" of the client-side application. The JPIP protocol is a more efficient service for interactive imaging, with fewer round-trip delays, and the opportunity for servers to efficiently manage their resources. JPIP requests identify the client's spatial region of interest, resolution and image components of interest, allowing the server to determine the most appropriate response elements and to optimally sequence them. As we shall see, JPIP allows the server to adjust the sequence in which data is returned so as to minimize disk thrashing while optimizing the image quality available at the client. A single JPIP request is sufficient to obtain an arbitrary region of an image, at a selected size/resolution, so that JPIP requests can be embedded as static targets within HTML pages. For interactive applications, multiple JPIP requests can be issued and efficiently served within an interactive browsing session, allowing the server to avoid the delivery of redundant data. The JPIP protocol has features which commend its use for collaborative remote image interaction, for providing services on unreliable transports (e.g., wireless transmission), for flexible and domain-specific delivery of meta data, and for applications which involve low data-rate networks.

These are just some of the reasons behind the development of a specific JPIP protocol which requires JPIP-aware server and client utilities. The present document focuses almost exclusively on the JPIP protocol, as opposed to byte ranging.

Section 3 discusses the key philosophical and architectural principles behind JPIP. These are tightly coupled to the JPEG2000 compression algorithm, which is briefly reviewed in Section 2. Section 4 then provides a technical description of some of the key elements of the JPIP protocol. Section 6 indicates how JPIP has been designed to meet the needs of a range of different application and transport environments, and then Section 7 presents experimental evidence for the efficiency, responsiveness and usability of one particular JPIP implementation.

2. JPEG2000 CODE-STREAM ELEMENTS

JPEG2000 is based on the Discrete Wavelet Transform (DWT), together with Embedded Block Coding with Optimized Truncation (EBCOT), as illustrated in Figure 1a. D stages of DWT analysis, labeled d=1,2,…,D, decompose the image into 3D+1 subbands, labeled LH_d, HL_d, HH_d and LL_D. Each subband is partitioned into rectangular blocks, known as "code-blocks," each of which is independently coded into a finely embedded bit-stream. Truncating the embedded bit-stream associated with any given code-block has the effect of quantizing the samples in that block more coarsely. Each block of each subband may be independently truncated to any desired length.

By discarding code-blocks corresponding to the highest resolution detail subbands, and omitting the final stage of DWT synthesis, a half resolution image can be reconstructed from the remaining subbands. Dropping the next lower resolution leaves a quarter resolution image, and so forth. This property allows JPEG2000 content to be delivered in a manner which matches the user's display resolution. To provide for progressive refinement of image quality, JPEG2000 introduces a quality layer abstraction[2]. Each successive layer represents incremental contributions (possibly empty) from the embedded bit-streams of all code-blocks in the image. These contributions may be adjusted so as to maximize the perceived image quality, as a function of the amount of data received at each layer boundary. Moreover, the optimal layering is roughly independent of the resolution which is to be delivered.

Spatial random access is possible, because each code-block is associated with a limited spatial region and is coded independently. To facilitate this, JPEG2000 defines collections of spatially adjacent code-blocks, known as "precincts." We say that a subband belongs to image resolution LL_d if it contributes to the reconstruction of LL_d, but not to LL_{d+1}. Thus, resolution LL_D contains only one subband, while all other resolutions LL_d contain the three detail subbands LH_{d+1}, HL_{d+1} and HH_{d+1}. Each precinct on LL_d consists of code-blocks from the same spatial region, within the subbands which belong to image resolution LL_d. Figure 1b illustrates the relationship between precincts, resolutions and code-blocks. Note that each code-block in the image belongs to exactly one precinct. While the precincts may be interpreted as partitions of their respective image resolutions, DWT synthesis involves filters with finite support so that the *reconstructed image regions which are affected by one precinct's code-blocks will overlap with the reconstructed image regions which are affected by adjacent precincts' code-blocks*. Conversely, the set of precincts which affect a given image region generally depends on both the precinct dimensions and the support of the DWT synthesis filters.

Associated with each precinct is a corresponding data-stream, representing the embedded bit-streams from all code-

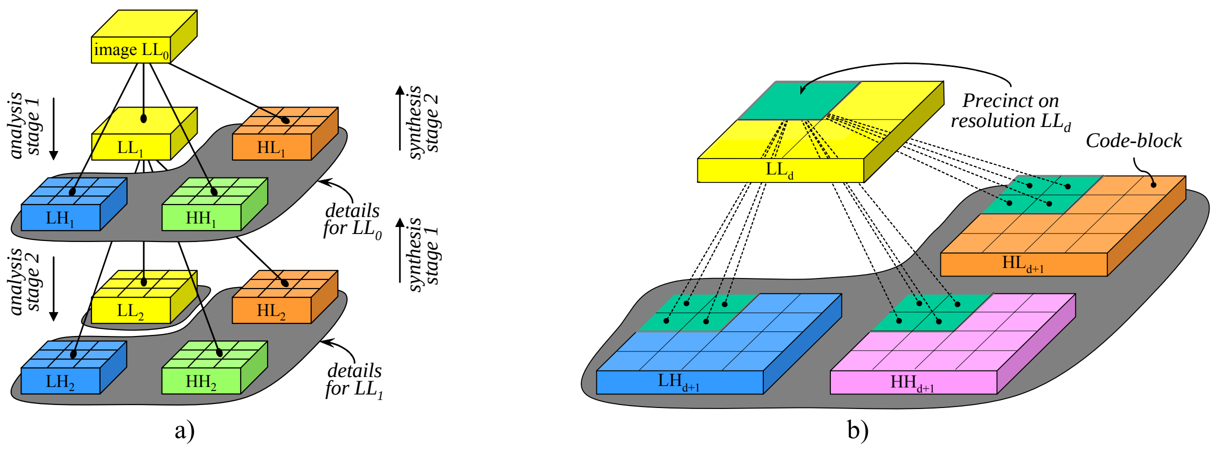

Figure 1. Elements of the JPEG2000 standard: a) DWT and code-block partition; b) precincts.

blocks in the precinct. As indicated in Figure 2, this data-stream is organized into segments, with one segment for each of the L quality layers. Each segment commences with a header, identifying the contribution made by each code-block's bit-stream to the corresponding quality layer. This is followed by the block contributions themselves. In a JPEG2000 code-stream, the segments are called "packets" and the packets from different precincts are interleaved following one of a number of predefined progression orders. For the purposes of JPIP, it is convenient to refer to the entire precinct data-stream as a single entity, any leading prefix of which may be delivered by the server.

3. ARCHITECTURAL PRINCIPLES OF THE JPIP PROTOCOL

3.1. Client-Server Interaction

Figure 3 illustrates the typical interaction envisaged by JPIP, between a client application and a remote server. For the purpose of this discussion, we shall assume that the client application is a graphical user interface, driven by an interactive user, although other applications can certainly be envisaged. We use the term "focus window" to refer to the user's current spatial region, resolution and image components of interest. This might only be a subset of the actual displayed image region, depending upon the application and user preferences. The window may also include other attributes, such as a maximum number of quality layers in which the user is interested. Typically, the interactive user pans (changes the spatial region) or zooms (changes the resolution or scale) the focus window.

The client contains a cache of the data previously transmitted by the server, organized into data-bins which are further explained in Section 3.2. The server may optionally maintain a model of the client's cache to avoid retransmission of data which the client already has; this is discussed further in Section 3.3. An important aspect of Figure 3 is the separation of the image decompression/rendering process from client-server communication. The inherent scalability of JPEG2000 means that an image can be meaningfully rendered from almost any subset of the original compressed data. This allows the focus window to be rendered directly from the cache before waiting for new data to arrive. In fact, it is often helpful to render a larger region of the image than just the focus window, to provide an interactive user with navigation context. As the client receives more data from the server, the cache contents grow and rendering is repeated to progressively refine the image quality both within and around the focus window.

The actual communication between client and server consists of request/response pairs. The request identifies the focus window via its geometric attributes, rather than low-level JPEG2000 constructs. This has several benefits: 1) JPIP requests can be compact and intuitive, facilitating their inclusion as URL's in HTML pages; 2) JPIP requests can be used to extract an appropriate image region from a non-JPEG2000 file (e.g. a server may offer a transcoding service); 3) the focus window expresses an end-user's ultimate interests, rather than a client's interpretation of those interests in terms of JPEG2000 elements allowing the server to determine how best to respond to the request.

The JPIP protocol is designed to be transport-neutral. A primary objective is that JPIP communication can be realized using HTTP/1.1 as the underlying transport, without interfering with existing HTTP infrastructure. However, JPIP can

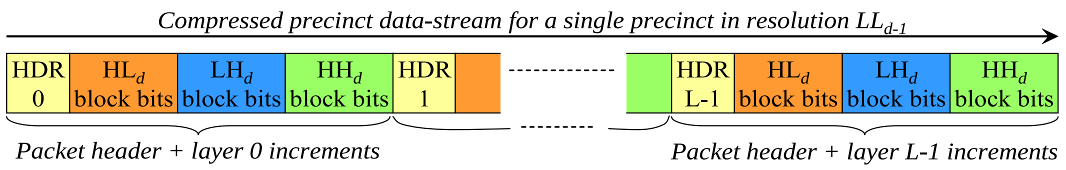
Compressed precinct data-stream for a single precinct in resolution LL_{d-1}
HDR 0
HL_d block bits
LH_d block bits
HH_d block bits
HDR 1
HDR L-1
HL_d block bits
LH_d block bits
HH_d block bits
Packet header + layer 0 increments
Packet header + layer L-1 increments

Figure 2. Precinct data-stream.

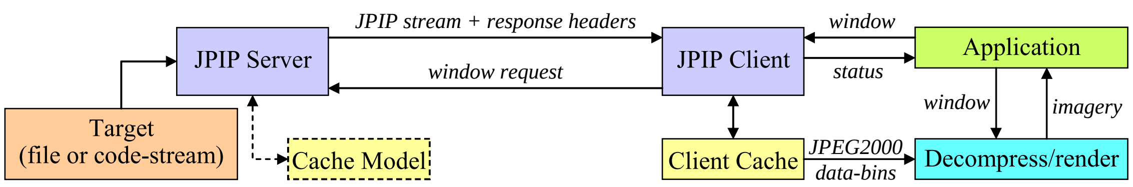

Figure 3. Client-server interaction in JPIP.

operate even more efficiently over other transports (see Section 6), and is well adapted to both reliable (e.g., TCP) and unreliable (e.g., UDP) transport environments.

3.2. Data-bins for All

JPIP defines a means for partitioning the semantic information within any JPEG2000 file into a collection of so-called "data-bins." While the protocol allows for out-of-order communication of any data-bin's contents to support unreliable transports, data-bins are typically transmitted and cached in linear fashion. For this reason, data-bins are designed to correspond with image objects that have a natural linear organization.

JPIP defines two partitioning schemes for JPEG2000 code-streams, based on either precincts or tiles as the predominant data-bins. In both cases, the code-stream's main header is assigned its own data-bin. In the case of precincts, each tile header is assigned its own data-bin and each precinct is assigned a data-bin which contains the precinct data-stream shown in Figure 2. In the case of tiles, each tile is assigned a single data-bin which represents the stream of data formed by concatenating all tile-parts of the tile, including all their headers. The tile-based approach offers server simplicity at the expense of reduced flexibility in the order in which data can be transmitted. The tile-based approach also relies exclusively on tiles to provide spatial accessibility, while the precinct approach offers spatial accessibility even if the entire image is compressed as a single tile. For the remainder of this paper, we focus exclusively on the more flexible precinct-based data-bins. For simplicity, we also ignore the possibility that the image may be tiled. For efficient interactive delivery, precincts should be as small as possible, containing at most 3 code-blocks, one from each subband. If necessary, the server can easily transcode the original code-stream to one with these minimal-sized precincts but using the same code-blocks as before; this process requires relatively little processing and is entirely lossless.

JPIP adopts a consistent approach to both image code-streams and meta data. JPEG2000 defines a family of file formats which may be used to encapsulate code-streams, all of which are based on a consistent "box" structure. Existing members of this family are JP2 (simple wrapper for a single JPEG2000 code-stream, defining color space and other rendering properties); JPX (extended version of JP2 with support for multiple code-streams, advanced color spaces, animation, etc.); MJ2 (motion file format, with timing information and one code-stream per video frame); and JPM (compound document file format). We use the term "J2 box" for the boxes found in these various files. A J2 box may be a super-box if its contents consist of a sequence of other J2 boxes (its sub-boxes). For the purpose of this document, the term meta data is used to refer to any J2 box (at the risk of some confusion, since code-streams are themselves embedded either by containment or by reference within J2 boxes).

Since all data communicated by JPIP must be in data-bins, a natural strategy would be to associate each J2 box with a separate data-bin. However, to preserve the interpretation of meta data in JPEG2000 family files, it is necessary to maintain the containment relationship of sub-boxes within their super-boxes, the relative ordering of J2 boxes, and even the absolute locations of J2 boxes within the original file. Absolute locations are required since certain J2 boxes can reference others through file offsets. To map the strongly file-oriented J2 box concepts into data-bins, JPIP adopts a simple yet elegant strategy. Conceptually the entire file is mapped to a root meta data-bin 0. Any top-level J2 box in a meta data-bin can be replaced by a special JPIP-defined "placeholder" box, which records the original box header details, but points to another meta data-bin for the contents of the box. If that box was a super-box, its contents are J2 boxes, any of which may similarly be replaced by a placeholder.

Placeholders allow J2 boxes to be partitioned into data-bins in any of a number of ways, with different implications for transport efficiency. The server makes the decision as to how data-bins should be constructed and served to the client. This allows for naïve servers which can only support simple files effectively, through to highly intelligent application and content aware servers. JPIP extends the placeholder concept by allowing any original J2 box to be replaced by an alternate "stream equivalent" box. Stream equivalent boxes may not have existed in the original file, but are better adapted for interactive delivery. This mechanism also allows the server to conceal references to other files which may be embedded within the meta data of the main file. Finally, placeholders may be used to indicate that a code-stream which was embedded in a J2 box is now available through the equivalent JPIP code-stream data-bins (precinct and header data-bins mentioned above). In the case of MJ2 files, the chunk-offset box (a large table of pointers used to index frames) may be replaced by a placeholder which efficiently identifies all code-stream data-bins associated with the frames in an entire video track. These facilities allow client rendering applications to work with cached JPEG2000 family files, exactly as though they were local files, in regard to both image content and meta data, while allowing file-centric concepts to be replaced by streaming-centric concepts where appropriate.

3.3. Caching and Sessions

As suggested by Figure 3, the client typically caches the data-bin contents transferred by the server in response to previous requests. JPIP identifies each request as either stateless or stateful. Stateful requests are made in the context of a communication session, whose state is maintained by the server. Although sessions require the server to allocate some persistent resources, sessions can significantly reduce its computational and I/O load. In the worst case, a stateless server may need to open the image, interpreting, extracting and reordering its contents each time a request arrives. On the other hand, a stateful server can generally minimize its interaction with the original image file, relying on information accumulated while processing previous requests.

Perhaps even more significantly, a stateful server may keep track of the number of bytes from each data-bin which it has already sent to the client. In this way, only the missing information need be sent in response to future requests. We refer to this as "cache modeling." For stateless requests, the client should explicitly identify the amount of data from each relevant data-bins which it already has in its cache, in each separate request. This tends to make stateless requests much larger. On the other hand, if a client cannot cache all information it has received from the server, stateful requests must explicitly identify the elements which have previously been received, but have since discarded from the cache. This is because JPIP invites all stateful servers to assume that the client caches all received data, unless otherwise notified. Cache model manipulation is discussed further in Section 4.4.

Clients may re-use their cache contents both within a session and between sessions. In fact, a stateless request is essentially just a session which terminates after the request has been serviced; it is obvious that clients must be able to re-use their cache contents between stateless requests. Moreover, clients may share their cache contents with others. For these reasons, JPIP servers must agree to maintain the integrity of each data-bin's contents across multiple requests, sessions and clients. To facilitate this, JPIP allows servers to assign a unique target-id to each target file or code-stream. If the contents of any data-bin associated with the target change (e.g., due to image editing), a new target-id must be issued to alert clients to the fact that their previous cache contents should no longer be considered valid.

3.4. Server Privileges

JPIP deliberately offers considerable flexibility in the way servers may respond to client requests. The server is free to transcode the original code-stream to one with smaller precincts, or one without any bulky pointer marker segments (since these are useful only when directly accessing the file), so long as it always presents the material in the same way (consistency of JPIP data-bins) for the same target-id. As noted in Section 3.2, the server is free to partition meta data (J2 boxes) in the original file into meta data-bins, following whatever strategy it deems best. Again, it must ensure that the target file is always presented in the same way when associated with a given target-id.

The set of code-stream data-bins which are relevant to any particular request is a deterministic function of the request parameters and the data-bin organization, which the server has selected to associate with the target file. However, the server has the flexibility to deliver the contents of these data-bins incrementally, following any order it chooses. For instance, some servers may employ content- and request-dependent rate-distortion optimization strategies to maximize the end-user's perceived image quality at each point in the transmission (explored in Section 7). In a distributed environment, the client's immediate server may itself have part of the image content, in which case it may choose to send this information first, so as to avoid delays while it attempts to recover further information from other servers (explored in Section 6).

Servers have potentially even more freedom in how they deliver meta data in response to a request. While JPIP defines request parameters that can be used to specifically identify meta data of interest, the client need not be explicit; it can ask the server to send any meta data which it believes to be relevant to the request's focus window. Consider, for example, a map service in which imagery is overlaid with scale sensitive meta data such as the names of regions, streets and buildings. The server may be able to associate meta data with a spatial region and prioritize the meta data according to the focus window's resolution. This can improve the user's interactive browsing experience by respecting the scale dependencies of map feature annotations.

Evidently, JPIP requests are not "idempotent," meaning that the response to a request is not necessarily the same each time the request is posed. In fact, unless the client explicitly indicates otherwise, JPIP servers are generally encouraged to interrupt the response to a current request when a new request arrives, since this would typically indicate that an interactive user's interests have changed. Exactly when the server chooses to do this is an implementation consideration. To minimize context-switching overhead, servers might choose to consider new requests only periodically, since client-side interaction can generate numerous requests as the user navigates through the image.

An important JPIP principle is that servers should be permitted to modify client requests. To understand why this is important, consider the problems which could be faced by a server trying to deliver progressively improving image quality over a very large focus window. If the precinct data-stream was organized in quality progressive fashion already on the server's disk, the server would incur severe disk thrashing as it attempted to serve small windows, since each increment in quality would require data from a different location on the disk. For this reason, each precinct data-stream is likely to have been stored contiguously within the original file. But then serving a large focus window with progressive quality refinement requires that either the server reads the relevant precinct data-streams repeatedly, or else it must pre-load them into memory. The first solution incurs excessive disk I/O, while the latter incurs excessive memory consumption. In an interactive setting it is practical for the server to explicitly reduce the resolution, region size or number of components in a large requested focus window. Interactive users can then readily learn how to best exploit the server's resources. In general, servers are permitted to reduce the scope of the requested focus window for either of the following reasons: 1) the request cannot be satisfied in quality progressive fashion without committing excessive resources; or 2) the requested window parameters do not correspond to a valid resolution, spatial region, components, etc. In either event, all modified request parameters must be signaled back to the client via response headers.

4. TECHNICAL DESCRIPTION

4.1. JPIP Streams and Messages

As mentioned in Section 3.2, the entire contents of a JPEG2000 file are partitioned into data-bins, including code-stream and meta data. To describe the server's response data, JPIP defines a new media type which we shall call a "JPIP stream." There are in fact two different media types, with tentative MIME types of "image/jpp-stream" and "image/jpt-stream." The former is used where the code-stream has been partitioned into data-bins based on precincts, while the latter is used for tile-oriented data-bins. In this paper, we shall collectively refer to these as JPIP streams.

A JPIP stream consists of a sequence of JPIP messages, each of which contains a range of bytes from the contents of a single data-bin. Each message has its own byte-aligned header which compactly identifies the data-bin class (meta data, code-stream main header, tile header, precinct or tile), a unique identifier for the data-bin within its class and the location and length of the range of bytes which may be found in the message body. Message headers may range from 3 bytes to more than 10 bytes in length, with shorter headers for more common or smaller messages. Transport efficiency is considered further in Section 7.

Since each message is fully self-contained, individual messages may be lost without necessarily affecting the usefulness of other messages in the stream. If the underlying transport protocol is unreliable (e.g., UDP), individual transport packets may be lost or misordered without adversely affecting the data contained in other packets, so long as each transport packet contains a whole number of JPIP messages. This can be achieved regardless of transport packet size constraints, since each message represents an arbitrary byte range from its data-bin.

It is worth stressing the fact that JPIP streams are image media in their own right. The stream of messages sent by the server in response to a JPIP request may be stored in a file and meaningfully opened as an image, without any reference to the request which generated the stream, or other server response headers. As such, a JPIP stream is essentially an alternate embodiment of the JPEG2000 compressed data. This alternate embodiment, however, has several desirable properties which are not possessed by JPEG2000 code-streams or files. JPIP streams can always be concatenated to form a new stream. JPIP stream messages may appear in any order and may contain overlapping byte ranges from the same data-bin. The state of a client's cache can always be represented as a JPIP stream and exchanged with other clients by any suitable means.

4.2. Basic Window Requests and Responses

JPIP requests consist of a sequence of "name=value" pairs. When carried by a text-oriented transport protocol such as HTTP, the name and value components of each request field are ASCII strings and request fields are separated by the

'&' character. This convention allows the entire JPIP request to be embedded in the query component (after the '?' character) of a conventional HTTP GET request, and hence captured by a URL which might be embedded in an HTML page. We note, however, that alternate representations may be preferred for other transports.

A basic JPIP request should contain some means for identifying the target file, together with the focus window. As an example, the following request is for a target file known to the server as "images/huge.jp2", at a resolution whose full image size is 12000x8000 (width by height), within a region of size 600x400, at an offset of 5000 pixels from the left and 6000 pixels from the top of the requested image resolution.

```
target=images/huge.jp2&fsiz=12000,8000&rsiz=600,400&roff=5000,6000
```

This request might be embedded in a URL such as

```
http://www.unsw.edu.au/bin/jpip.cgi?target=images/huge.jp2&fsiz=12000,8000&rsiz=600,4
00&roff=5000,6000
```

The target might be conveyed by other means, outside the scope of the JPIP protocol itself, as in the following example where the target file is identified by the resource component of the URL:

```
http://www.unsw.edu.au/images/huge.jp2&fsiz=12000,8000&rsiz=600,400&roff=5000,6000
```

The target may also be a byte range of a file which is known to include JPEG2000 content (e.g., a PDF file). Finally, the target might be implicitly identified by any request which is issued within a stateful session (see Section 4.3).

Recall from Section 2 that the DWT used in JPEG2000 provides only D+1 different image resolutions, whose dimensions W_dxH_d are smaller than those of the original image by 2^d, where $0 \leq d \leq D$ and D is the number of DWT levels. If the dimensions supplied by the `fsiz` (full size) query field are not identical to those of one of these image resolutions, the JPIP server is expected to round to the nearest available resolution, W_dxH_d. The rounding direction may optionally be made explicit using a request field of the form "`fsiz=12000,8000,round-up`" or "`fsiz=12000,8000,round-down`" or "`fsiz=12000,8000,closest`". Wherever this happens, the region size and offset parameters supplied by the `rsiz` and `roff` query fields must be scaled so as to preserve their relationship to the `fsiz` parameters.

In view of the above discussion, it is evident that the server may sometimes need to modify request parameters. In Section 3.4 we provided other examples of situations in which the server is at liberty to modify request parameters. As noted there, an important feature of JPIP is the freedom given to servers to respond to a request after first modifying the parameters. In recognition of this feature, JPIP defines syntax for servers to signal the changes which have been made together with their response. In fact, JPIP requires that servers explicitly signal all changes to request parameters via "response headers." If HTTP is used as the transport, JPIP response headers should be included along with the other HTTP response headers prior to the message body, using header names which are constructed by prepending the name of the modified request field with the string "`JPIP-`". The example below illustrates how the server might respond to an HTTP GET request for an image resolution which differs from the nearest available one.

```
← GET /images/huge.jp2&fsiz=12000,8000,round-down&=600,400&roff=5000,6000 HTTP/1.1
← Host: dst-m
    → HTTP/1.1 200 OK
    → JPIP-fsiz: 6000,8000
    → JPIP-rsiz: 300,200
    → JPIP-roff: 2500,3000
    → Content-type: image/jpp-stream
    … (rest of the response)
```

Following any response headers, the server sends a sequence of JPIP messages. Thus, unless no response data is required, the JPIP server's response always includes a valid JPIP stream. As noted in Section 3.4, the server may choose to truncate this stream at any desired point if a new request arrives, allowing it to process the new request in a timely manner. However, each response must be terminated by a special EOR (End of Response) message, which includes a code identifying the reason for termination. Using the EOR message, a client may readily determine whether the request has been terminated because all relevant data has been received, or for another reason such as the arrival of a subsequent request. Although the EOR message uses similar constructs to JPIP stream messages, it is not actually part of the JPIP stream.

Although JPIP streams are the most natural response data type for JPIP client/server applications, it is also possible for clients to request that the response be converted into a standalone image such as a JPEG image or a standalone JPEG2000 image. This is done by including a `type` request field, as in

```
target=huge.jp2&fsiz=1280,1024&rsiz=300,200&roff=100,80&type=jpeg,jp2
```

which means that the client is prepared to receive the requested region as a complete image, having a file format whose MIME type is one of "image/jpeg" or "image/jp2". Of course, servers might not offer such a transcoding functionality. Also, many JPIP concepts and request fields are meaningful only when the response is a JPIP stream.

4.3. Channels and Sessions

As noted already, requests may be either stateless or stateful. A stateful request is issued in the context of a session, which the server manages on the client's behalf. It is actually possible for the client to open multiple channels to a single session, allowing it to issue multiple requests concurrently. The JPIP streams returned in response to requests on each channel may be combined in any order to form the evolving response. This allows sophisticated clients to pose rich requests, consisting of multiple focus windows. Each channel has a server-assigned channel-id string, which must be included in any session-oriented request via the `cid` request field. The channel-id implicitly identifies the session, as well as other invariants of the session such as the target file and previously signaled client preferences.

Channels and sessions are established with the aid of the `cnew` request field, and the server's `JPIP-cnew` response header, using a boot-strapping procedure in which an initial stateless request is used to obtain the first channel of a new session. To illustrate the procedure, consider the following example.

```
← target=image.jp2&fsiz=4096,4096&rsiz=512,512&roff=1300,1700&cnew=http
    → JPIP-cnew: cid=01ab7f,transport=http,host=107.41.39.1
    … (rest of the response to original request, delivered as though request had been issued with cid=01ab7f)
← cid=01ab7f&fsiz=4096,4096&rsiz=512,512&roff=1800,1900
    … (response to request)
← cid=01ab7f&fsiz=4096,4096&rsiz=512,512&roff=2100,2000&cnew=http
    → JPIP-cnew: cid=3c0081,transport=http
    … (response to request)
```

There are several points to observe from this example. Firstly, the `cnew` request field identifies the transport protocol to be used during communication with that channel. In the example, this is HTTP, but JPIP defines other transports and allows for future adaptation to just about any underlying transport mechanism. In fact, each channel can use a different transport. Secondly, the server's response identifies the new channel-id, the transport to be used, and potentially other transport-specific information. In this case, the `JPIP-cnew` response header identifies the IP address of the machine which is hosting the session. This might be different from the host which responded to the initial stateless request if redirection is required to find a server willing to host the session. Finally, observe that new channels may be opened to an existing session by including a `cnew` field within a request which is already associated with that session.

JPIP provides a `cclose` request field to be used when issuing the final request within any given channel. This provides a clean mechanism for the server to efficiently recover the channel's resources. The use of `cclose` does not inherently imply that the underlying transport protocol's connection should be closed. For example, if HTTP/1.1 is used as the transport, a persistent TCP connection will generally remain open until a "`connection: close`" header is sent by the client. This allows further requests to be issued without re-establishing the TCP connection. In fact, it is quite legal to issue requests with different JPIP channel-ids on the same transport connection.

4.4. Cache Management Instructions

Where requests are issued within the context of a stateful session, it is generally assumed that the server will maintain a model of the client's cache contents, so that only new data need actually be sent in response to any request. Under some circumstances, however, the client may need or wish to explicitly modify the server's cache model. If the client has cached data from a previous browsing session or from previous stateless requests, it may wish to add the corresponding elements to the server's cache model so as to avoid redundant transmission. On the other hand, if the client has limited cache memory, it may need to delete some elements from its cache; these elements should also be removed from the server's cache model so that they can be re-transmitted if the client requires them at a later point.

JPIP's `model` request field may be used to explicitly add or subtract elements from the server's cache model. In particular, the client may request that the server add either the entire contents of any data-bin, or a leading prefix of any data-bin to its cache model. The client may also request that the server remove either the entire contents, or all but a leading prefix of any data-bin from its model. For example, the following request field asks the server to add the main and tile headers, meta data-bin 0, and the first 20 bytes of precinct data-bin 0 to its cache model, and to remove precinct data-bin 1001 from its cache model.

```
    model=Hm,H*,M0,P0:20,-P1001
```

At first glance, it may appear as though the client could place unreasonable demands on the server's memory resources by expecting to directly manipulate the server's internal cache model. However, JPIP servers are not obliged to maintain a complete model of the client's cache. For example, the server may arbitrarily discard elements from its cache model if it is running low on memory. In the extreme case, a server might choose not to model anything, in which case it can entirely disregard all cache model statements. Of course, this will generally result in less efficient communication, since the client may receive copies of data which it already has in its cache, but the protocol still works. It is also worth noting that there is no need for the client to inform the server of the state of its cache, except in relation to those data-bins which are actually relevant to the requested focus window.

For stateless requests, a typical client will identify the contents of its cache within each request. Conceptually, each stateless request initiates a new session on the server, whose cache model is initially empty. The model is manipulated by any cache management instructions in the request, after which the focus window is processed and appropriate non-redundant return-data is delivered. The session is then destroyed at the end of the request. JPIP defines a special syntax for efficiently performing the cache management interactions which are relevant to stateless requests.

4.5. Other Features

Beyond the basic features mentioned above, the JPIP request syntax provides many additional features. Each request may be qualified with a byte limit, identifying the maximum number of JPIP message body bytes to be returned in response to the request. This allows clients to maintain responsiveness by preventing the server from "flooding" network buffers in response to an individual request. The number of code-stream quality layers may be similarly restricted, and an abstract image quality limit may also be imposed. Although the default behavior is for new requests to pre-empt the server's response to a current request, a new request may explicitly specify that the server should wait until the previous response is completed before processing the new request.

In addition to explicit focus windows, JPIP allows the client to request a specific region of interest whose details are only known to the server or are identified within the target file itself. Where the target file involves multiple code-streams, the request may explicitly identify one or more code-streams which are of interest. Intelligent clients may explicitly restrict or augment the types of meta-data which the server is being asked to return in response to the request. Clients may also signal general preferences, which intelligent servers can use to customize their responses.

5. BYTE RANGING

HTTP/1.1 allows clients to make byte-range requests into a target file. Aspects of the functionality of the JPIP system described above can be achieved using byte-range requests in conjunction with JPEG2000 files. While JPEG2000 code-streams may optionally include pointer information which could be used to formulate appropriate byte-range requests, the pointer information itself may be very large and is not amenable to random access. To facilitate efficient access, based on byte-range requests, the JPIP standard describes the form of a new set of index tables, which may be included in JP2 family files. Two important points to be made initial are: 1) the motivation for defining index tables in JPEG2000 Part 9 is to permit early implementation of some of the JPIP functionality on current internet infrastructure; and 2) using byte-range requests to achieve JPIP functionality is not a JPIP protocol – byte-range requests are already part of HTTP. JPIP index tables provide clients with a tree-structured, randomly accessible catalog of the byte-ranges associated with header information, codestreams, tile-parts, precinct packets and meta data boxes. After using the index tables to identify and access important header information, a client can determine what is needed to satisfy the needs of the client-side application, making further requests for the relevant index information and ultimately the relevant data. Byte-range requests require no special JPIP image server, and are compatible with the caching infrastructure provided by HTTP/1.1.

There are two significant disadvantages of the byte-ranging approach. First, the client must make a number of round-trips to selectively retrieve both index information and image data. Secondly, data transfers will usually be sub-optimal in the sense of improving perceived image quality as quickly as possible. One reason for this is that clients typically do not have sufficient information about the contribution of each byte-range towards image quality (or information quality, in the case of meta data). In general, only the server can balance the information priorities of the client with other priorities such as access cost, transmission cost, and server load considerations. As noted previously, the server might not have local access to the entire image, which is something the client cannot generally know. In contrast to byte-ranging, the JPIP protocol allows servers to exploit their knowledge of the image quality implications for data which they serve. It even allows servers to generate the compressed imagery dynamically, in response to client requests.

6. DEPLOYMENT

So far this paper has alluded to underlying assumptions on the requirements of typical JPIP systems, and the philosophy of how they would be met. The purpose of this section is to review these issues by describing common JPIP deployments. At the most basic level, an everyday use-case would be a user with some GUI application, navigating an arbitrary large (e.g. geospatial) image over a network. Some present day paradigms would see the entire imagery downloaded to the client before it was available for exploitation, but this is not practicable on the internet. An alternative approach is for the server to create and send a "stand-alone" image (e.g. a JPEG image) based on the client's focus window request. This is commonly called "screen-scraping" and indeed it is envisaged that JPIP image servers may offer a transcoding service (e.g. via "`type=jpeg`" requests) as a means of permitting backward compatibility for non-JPEG2000 capable clients. However, screen-scraping is not smart dissemination as each stand-alone response has limited reuse potential. Since JPEG2000 has a number of scalable dimensions with randomly accessible data, JPIP is designed to serve this data. Smart dissemination is achieved through the client caching the received JPIP stream, the server keeping a cache model (assumes that all JPIP stream data is cached unless otherwise signaled), and the server only transmitting the difference between the set of data required for the current focus window request, less the intersection of this set with what is in the cache model.

A typical model for interactive JPIP communication has already been described in Section 3.1, in connection with Figure 3. An initial request would typically identify a low resolution focus window, providing the interactive user with sufficient detail to commence meaningful navigation within the image. Responding to the demands of an interactive user, the client may generate requests at an arbitrary rate. A well-behaved server would process these requests in a manner that provides favorable responsiveness. If the server takes a period of time to set-up for a JPIP response and to transmit some "meaningful" amount of data, it makes no sense to respond to requests at a faster rate. What defines the amount of data that would be "meaningful" is application specific, but the server should only respond to the most recent request sampled from the request stream at no more than this rate. A formal response must be issued to each request, but the response can be empty if the server has a more recent request in the queue. Thus the server will often provide empty responses to requests. Of course, this does not prevent the application from rendering the relevant imagery using the contents of the client-side cache. This behaviour implies that client requests are pre-emptable, meaning they are not guaranteed to result in complete responses. In fact, there is no way to issue a JPIP request which is not pre-emptable. It is possible, however, to issue a request which will not pre-empt a previous request; this is done by including a "`wait=yes`" field. Even though requests are pre-emptable, the client can know whether it has all image data relevant to a particular request, based on the reason code included in the EOR message which the server appends to each response.

Pre-emptable requests are an example of a core JPIP philosophy to couple client-server interaction as loosely as possible. Since the client could change the request parameters and/or cache model at any time (via a new request), tight server-client synchronization is problematic. Another example where tight coupling is undesirable is for unreliable transports, where network-data packets (as opposed to JPEG2000 packets) could be received out of order, delayed, or not arrive at all. Such environments are common in wireless networks, particularly in military wireless networks. The recommended practice for military applications is to utilize UDP as the transport[3]. Since JPIP imagery data-bins correspond to embedded bit-streams, each contiguous prefix can be decoded to yield image quality improvements, regardless of what other data may have been delayed or lost in the network. For unreliable transports, relatively little hand-shaking is required to manage the integrity of the server's cache model. In particular, it is generally sufficient for the server to know which network packets might have been lost in transit. If the client intends to explicitly manipulate the server's cache model, some additional signalling is required to ensure proper ordering of requests at the server.

Interactive applications are generally expected to remain responsive to changes in the client's interests. Pre-emptive handling of client requests goes part way to maintaining responsiveness, but responsiveness can also be hindered by intermediate buffering of server response data within the network. If neither the client nor the server takes steps to implement flow-control management, the server's response to a previous request may be queued ahead of a slow link and block responses to future requests from arriving in a timely manner. Section 7.2 compares the impact of client-based flow-control and server-based flow control. When using HTTP as the transport, client-based flow control is the only practical option, requiring that the client explicitly limit the amount of data which the server can return in response to a request. Using a different transport defined by the JPIP standard, however, allows server-based flow control which results in better utilization of the channel. Server-based flow control is another example of weak coupling between client and server, since the way in which a response is answered is not determined exclusively by the client.

An excellent example of the advantages potentially offered by weakly coupling the server's response with the client's request is a system of cascading JPIP servers, performing the functions of network proxies and caches. Consider a number of users connected to a local JPIP server that initially holds no imagery data. The local JPIP server acts as a proxy for a remote server. The local server simultaneously delivers JPIP requests to the remote server and answers JPIP requests from its clients, caching JPIP stream responses from the remote server and using its cache to answer and optimally sequence JPIP stream responses to the client. The local server does not need to download the entire image in order to serve clients. Note that the JPIP proxy server's cache would be persistent between sessions. Then, if a second client interrogates the same imagery, the local JPIP server can respond out of its own cache to the extent that the second client's requests overlap those posed previously by the first client. Meanwhile, the local proxy server interrogates the remote server to recover data which is still missing. In this way, the order in which data is delivered in response to client requests may be highly dependent on the distributed service environment, and the way in which it has been used by other clients.

To demonstrate the flexibility of the JPIP system further, notice that the remote server could be a simple file server with no JPIP capabilities. In this case, the local JPIP proxy server would use JPIP index tables to issue byte-range requests to the remote server, while providing an efficient communication with the local client via the JPIP protocol. Such an architecture has much to recommend it. High level focus window requests received by the local server via the JPIP protocol enable it to predict and pre-fetch the data which is most likely to be required from the remote server, rather than simply passing on byte-range requests from the local client. On the other hand, interacting with the remote server via byte-range requests allows existing caching infrastructure developed for HTTP to be fully utilized in the wide area network.

JPIP's loose coupling philosophy for imagery is also applied to meta data. As a general rule, it is easier for the server to understand the potentially large and complex catalogue of meta data in a JPEG2000 file. The server is at liberty to compress the catalogue using placeholders and equivalence redirections. This does not restrict the client's ability to access meta data; rather, it should actually reduce the average number of request-response round-trips required. JPIP provides interactivity with motion JPEG2000 files (MJ2) and compound documents (JPM), and it is intended to be compatible with new parts of the JPEG2000 standard which are being develoed to support scientific 3-D imagery (JP3D), wireless applications (JPWL) and security (JPSEC). We envisage future intelligent JPIP servers which will utilize imagery importance maps[4, 5] to prioritize data for JPIP streams in a scalable fashion. These are also likely to have applications in wireless video. A JPIP service to memory challenged clients is achievable using the cache management signaling in JPIP. While this would be somewhere between full caching and screen-scraping, depending

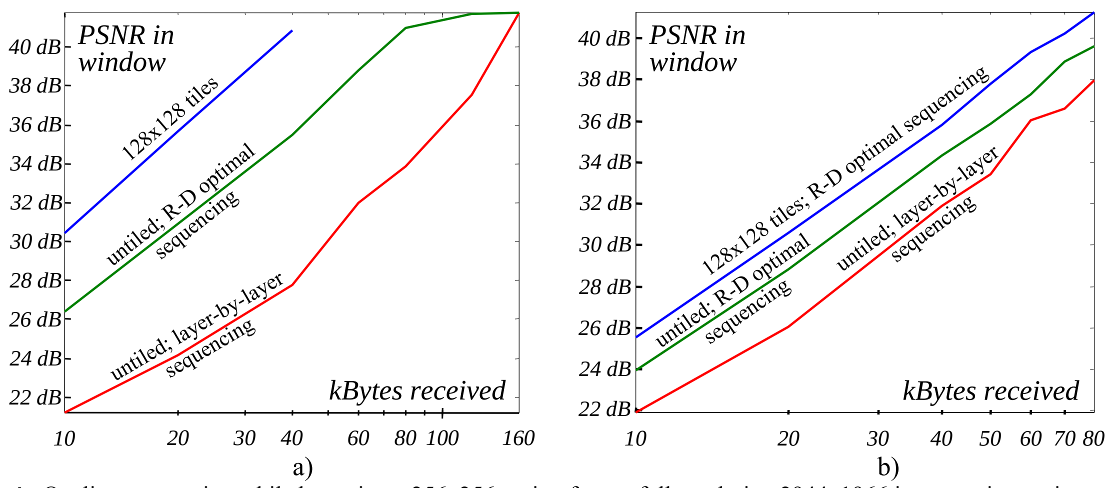

Figure 4. Quality progression while browsing a 256x256 region from a full resolution 2944x1966 image, using various service policies: a) region aligned at 1280x1024; b) region aligned at 1325x1063.

on the amount of memory available, it is another example of the flexibility offered by JPIP. With these few examples, we hope to have demonstrated that JPIP is capable of meeting the needs of a broad range of applications for interactive interrogation of JPEG2000 based data.

7. PERFORMANCE

7.1. Spatial Access Efficiency and the Benefits of Server-Driven Information Sequencing

In Section 2, we pointed out that the reconstructed image region which is affected by any given precinct overlaps with the reconstructed image region which is affected by neighboring precincts. This means that in order to fully reconstruct a given region, it is necessary for the server to deliver data for more precincts than one might at first expect. Moreover, if the precinct dimensions are the same at every resolution (typically 64x64 or 32x32 in our case), the precincts from lower resolutions will have a much larger region of influence within the reconstructed image than those from higher resolutions[*]. Again, this means that the precinct data which the server must deliver to accommodate a small spatial region of interest may actually represent a significantly larger region. It is natural, then, to ask how efficient the spatial random access offered by JPIP actually is.

Figure 4 plots the received image quality (PSNR) within a 256x256 focus window, as a function of the total number of bytes actually transmitted by the server in response to a request for that focus window. Three different service strategies are considered, with two different locations for the focus window. The lower curve in each plot corresponds to the delivery of precinct data in quality progressive fashion, following the quality layers in the original code-stream. The second curve is similar except that precinct data-bin contributions are optimally sequenced at the server so as to maximize the received image quality at each point in the progression. The server's rate-distortion optimization algorithm takes into account the degree to which each precinct contributes to the focus window, as well as operational rate-distortion information collected during compression. Details of the optimization algorithm are provided in[6]. Evidently, server-driven information sequencing has the potential to dramatically improve the transport efficiency.

The upper curve in each plot corresponds to the JPIP delivery of an image which has first been decomposed into independent tiles of size 128x128, each of which has been independently compressed. When the tiles align perfectly with the requested focus window, as they do in Figure 4a, no redundant data need be transmitted at all. The performance in this case is indicative of that which could be achieved if the requested focus window were extracted from the image and independently compressed and delivered to the receiver. Thus, the difference between the upper

[*] It is possible to work with precincts and hence code-blocks whose dimensions decrease from resolution to resolution so as to maintain a roughly constant region of influence in the original image, but this seriously compromises compression efficiency, especially when accessing the image at lower resolution.

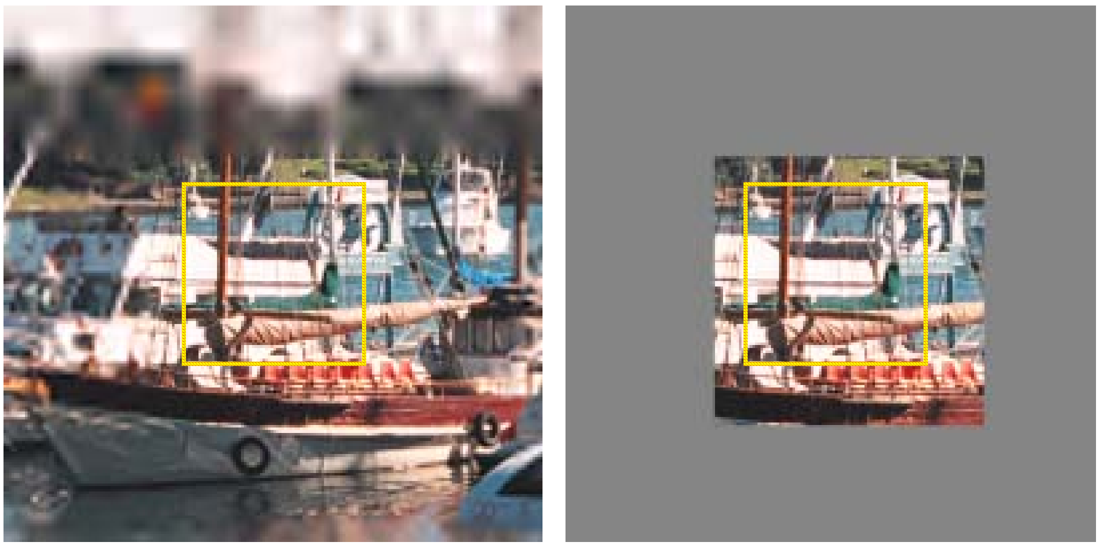

Figure 5. Image recovered after receiving all data for the 256x256 focus window identified by the box: a) where the image was untiled; b) where the image was partitioned into 128x128 tiles.

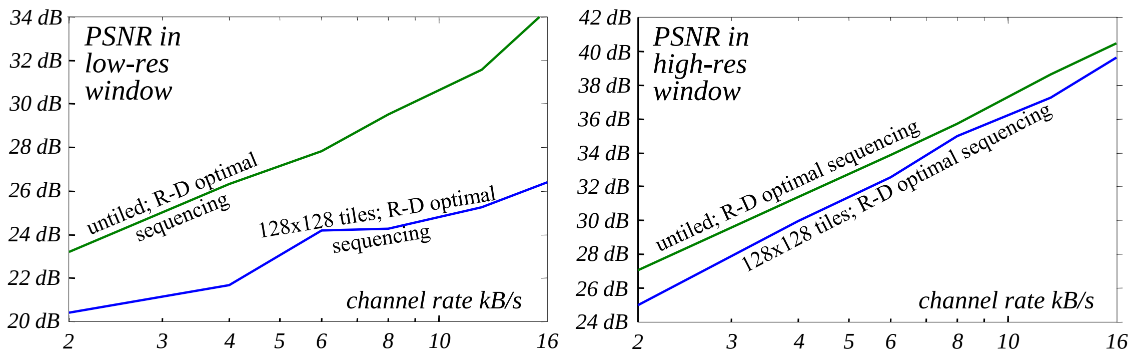

Figure 6. Quality progression during multi-resolution browsing with a 640x480 focus window: 5 seconds at quarter resolution, followed by 10 seconds at full resolution.

and middle curves in Figure 4a is indicative of the "random access cost" associated with serving a small spatial region from an image which has been compressed without tiling. This cost reduces as the focus window increases in size.

Tiled images are fully supported by JPEG2000 and JPIP, although the benefits of tiling clearly decrease if the tiles are not perfectly aligned with the region which is actually of interest, as demonstrated by Figure 4b. It can also be argued that untiled images provide much more navigation context for the interactive user, at a relatively small cost. To illustrate this point, Figure 5 shows the images recovered in response to the unaligned focus window request of Figure 4b. The tiled image is fully recovered with about 20% fewer received bytes than the untiled version, with identical quality in the focus window itself, but the untiled image provides cues regarding the surrounding regions.

Finally, we note that tiled images are not well suited to the efficient communication of images at reduced resolutions. To illustrate this point, Figure 6 shows results obtained from a more realistic browsing session, in which the 2944x1966 image is first accessed at quarter resolution (736x492) over a 640x480 focus window for a period of 5 seconds, and then at full resolution over a centered 640x480 focus window for a period of 10 seconds. Evidently the performance within both resolutions is improved by avoiding tiles altogether, relying instead on JPIP's precinct data-bins for spatial random access. The fundamental problem with tiles is that the number of tiles is independent of the resolution at which the image is to be accessed. As a results, tiles which start out with a reasonable size at the full image resolution can become very small at lower resolutions, where the efficiency of the compressed representation is correspondingly reduced[7].

7.2. Transport Efficiency and Responsiveness

In this section, we briefly examine the efficiency and responsiveness of the JPIP protocol itself. We do this by synthetically generating a consistent set of focus window changes at timed intervals, and monitoring the server's response traffic. All communication is throttled to a maximum rate of 4 kbytes/s and the focus window changes every 5 seconds, through a sequence of resolution changes and diagonal panning into a losslessly compressed color image of size 13000x13000 (compressed size is 220 Mbytes; original uncompressed size is about 500 Mbytes). Two different transports defined by the JPIP standard are considered in this experiment. Using HTTP/1.1 as a transport, the only way to maintain responsiveness to new requests is for the client to constrain the server's response length (max message body bytes) using JPIP's `len` request field. If the client does not do this, the server will tend to flood intermediate network buffers with its response to a request, preventing the client from receiving any response to a new focus window request in a timely fashion. The client dynamically estimates the channel conditions (delay and bandwidth) and issues overlapping length-constrained requests so as to use all available bandwidth while maintaining responsiveness. The client aims to keep response time within around 1 second by modulating the maximum response length identified via the `len` request field.

The second transport is known to JPIP as "http-tcp." It employs HTTP to pose requests and receive response headers, but all JPIP stream messages are returned to the client on a separate TCP channel. The client sends regular acknowledgement messages back to the server on this auxiliary TCP channel, allowing the server to estimate channel conditions (delay and bandwidth) and hence regulate the flow of its response data to maintain responsiveness. The benefits of this are that the client sends fewer requests, the server sends fewer response headers, and the server's packaging of response data into JPIP messages is not constrained by artificial byte lengths. In our experiments, the server aims to keep response time around 1 second, the same value selected for the client-driven HTTP transport.

Figure 7 (left) identifies the cumulative amount of response data received since the last change in focus window. Since the focus window changes every 5 seconds, we see that both transports are indeed able to maintain a response time of roughly 1 second, albeit by very different means. Figure 7 (right), however, clearly reveals the improvement in transport efficiency associated with the "http-tcp" transport. The signaling overhead reported here accounts for the size of server response headers as well as JPIP message headers, both of which are larger on average with HTTP as the transport. The figure plots cumulative signaling overhead, as a percentage of cumulative received data, since the last change in focus window, which explains the observed transitions roughly every 5 seconds. Evidently, the average signaling overhead for JPIP need not be larger than about 15%.

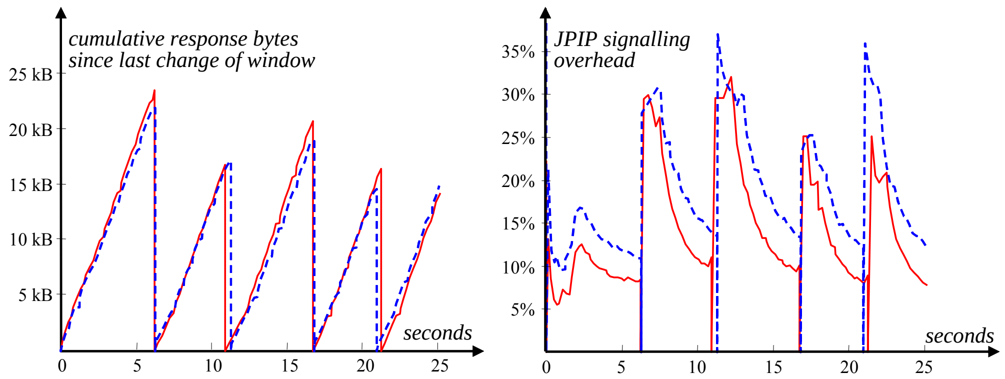

Figure 7. Responsiveness and signaling overhead of JPIP using two different transports: "http" – dashed line; and "http-tcp" – solid line.

7.3. Usability Observations

We have undertaken a performance trial of a JPIP client-server system[8]. The server was hosted by a P4 2.4GHz machine with 1 Gbyte of 333 MHz memory and a 100Mbps network port. Clients were located in capital cities around Australia via a corporate LAN. The server was configured to limit the outgoing data rate to either 8, 4 or 2 kbytes/s for each client. Over a 10 minute interval, about 20 clients on varying desktop PCs simultaneously and independently interrogated a satellite monochrome image measuring 84286×84286, whose original size was 6.6 Gbytes, losslessly compressed to 3.4 Gbytes. Users reported satisfaction with the 8 kbytes/s service, and were accepting of the 4 and even the 2 kbyte/s services. The average CPU usage on the server ranged from 5% to about 10%. Network activity peaked at 140 kbytes/s, or about 1% of the port capacity. Sustained disk access ranged from 1 Mbyte/s to 5 Mbyte/s (average about 3Mbyte/s). Memory usage was fairly linear at 30 Mbyte per client. Clearly, this type of machine could comfortably serve 30 clients before it was bounded by memory limits.

The large per-client memory usage observed in this experiment is a consequence of the fact that the server's cache model is currently implemented by allocating one 32-bit word for each precinct in the entire image, even though only a very small percentage of the precinct data-bins are ever accessed by any given client. Future implementations will address this issue through the use of more efficient data structures for the sparse cache model. It is worth mentioning, however, that JPIP servers are not obliged to maintain complete cache models. A server could maintain a bounded list of region-based cache modeling structures, for example, discarding information about regions which have not been accessed for some time. The only impact of such a resource constrained implementation would be that communication efficiency might be reduced for certain client access patterns. One clear conclusion from these experimental observations is that the server is not CPU bound. With more sophisticated memory structures for cache modeling, such a desktop machine could be expected to serve hundreds of clients.

8. CONCLUSIONS

This paper has described the philosophy and architecture of JPIP, together with the envisaged modes of operation and some preliminary performance observations. One of the key principles behind the development of JPIP is that server responses be coupled as loosely as possible to client requests. In fact, the server's response to any request is a self-describing JPIP stream, whose length and internal organization are at the discretion of the server. JPIP has been designed to provide efficient data transfer, responsive performance, flexibility and effective random access for JPEG2000 data. Its philosophy is consistent across all data types, including code-stream headers, compressed imagery data, and meta data. This allows JPIP to provide interactive services for all members of the JPEG2000 family of files, including JP2, JPX, MJP, JPM and later JP3D. Moreover, JPIP servers can be developed to efficiently meet the needs of a wide range of applications, from simple image browsing, to sophisticated multi-channel navigation of massive hyperspectral images overlayed with context-dependent meta data. The intention of this paper has been to emphasize the fundamental principles behind the JPIP standard, while at the same time providing a useful introduction to its architecture, its basic syntactic elements, and some anticipated deployment paradigms. JPIP is expected to play an important role in accelerating the adoption of JPEG2000 technology within a wide range of application domains, including medical, military, surveilance, mobile, and internet imaging.

Acknowledgement

Thanks to Michael Owen of DSTO for performing the usability trial.

9. REFERENCES

1. S. Deshpande and W. Zeng. Scalable Streaming of JPEG2000 Images Using Hypertext Transfer Protocol. *Proc. ACM, MM,* 372-281, 2001.
2. D. Taubman and M. Marcellin. *JPEG2000: Image Compression Fundamentals, Standards and Practice.* Kluwer Academic Publishers, Boston, 2002.
3. R. Prandolini, T. A. Au, A. K. Lui, M. J. Owen, and M. W. Grigg. Use of UDP for efficient imagery dissemination. *Int. Symp. Visual Communications and Image Processing (VCIP), Perth, Australia*. June 2000.
4. R. Prandolini. Coding of surveillance imagery for interpretability using local dimension estimates. *Int. Symp. Visual Communications and Image Processing (VCIP), Perth, Australia*. June 2000.
5. A. Nguyen, V. Chandran, S. Sridharan, and R. Prandolini. Importance Assignment to Regions in Surveillance Imagery to Aid Visual Examination and Interpretation of Compressed Images. *Int. Symp. Intelligent Multimedia, Video & Speech Processing, Hong Kong.* May 2001.
6. D. Taubman. Rate-distortion optimized interactive browsing of JPEG2000 images. *Proc. IEEE Int. Conf. Image Processing (ICIP), to appear.* September 2003.
7. D. Taubman. Remote browsing of JPEG2000 images. *Proc. IEEE Int. Conf. Image Processing (ICIP).* 1:229-232, September 2002.
8. D. Taubman. Proposal and Implementation of JPIP (Jpeg2000 Internet Protocol) in Kakadu V3.3. *via* http://www.kakadusoftware.com. August 2002.